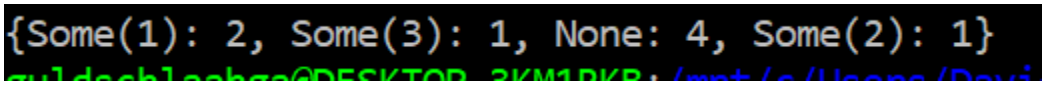
David Lau 253 Reuse distance report

Functionality

To test the program, I created a vector of integers: [4, 3, 4, 5, 5, 5, 6, 4]. This should return a hashmap from reuse distances to the number of times they occurred of 4 instances of infinity(None), 2 instances of 1, 1 of 3, and 1 of 2. Printing this hashmap displays the following:



Showing that the program works correctly.

Complexity

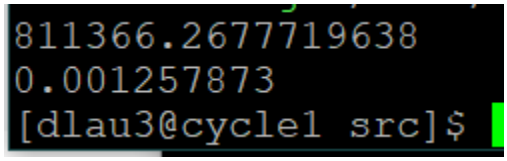
The time complexity will be O(nm) by the following analysis: The program will loop n times. For every loop, the program will add the current element to all the hash sets in the hashmap, which will take O(m) time. So the run time is O(n*m).

The space complexity will be O(n + m^2). N slots will be filled by the input, and the values are stored in a hashmap of key type of the input element, into hash sets of the same element type, which has m variants, so m hash sets of size m can be stored.

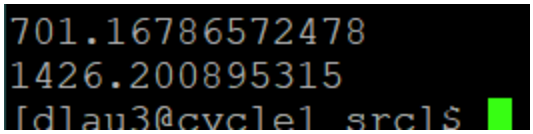
Speed Tests

Top value is reuses per second, bottom is number of seconds to run total. The tests were ran with cargo run --release.

10^3:



10^6:

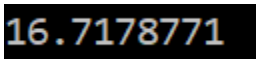


10^9:
Ran for longer than an hour.

Analysis

The runtime is almost exclusively determined by the complexity. This is because essentially all the operations used are O(1) with the exception of the costly hashset.clear function, and mostly manipulating pointers. To show this, examples of the time with a given n or m along with their resulting time, along with the same for 2n and 2m, were tested and the results shown to show how they affected the time.

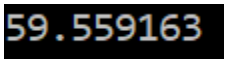
n, m:



2n, m:

n, 2m:

2n, 2m:



This demonstrates the essentially proportional effect. The doubled input seems to result in a slightly larger increase than the doubled key number, which I am unaware of the cause of.